Sokol Saliu

Design of a microcontroller based system The MC68HC11

GOTHENBURG 2003

Version 03/08/27

Contents

1 Building a microcontroller system **5**
- 1.1 Design . . . 5
- 1.2 Project . . . 5
- 1.3 The MC68HC11 microcontroller family . . . 6
- 1.4 Development tools . . . 7

2 Implementation of a complete single-chip system **7**
- 2.1 Interfacing SCI with a terminal . . . 10

3 Debugging **12**
- 3.1 Debugging methods . . . 12

4 A MCS in expanded mode **13**
- 4.1 Bus demultiplexing . . . 13
- 4.2 Timing diagrams . . . 13
- 4.3 Tristate bus . . . 15
- 4.4 Address Decoding . . . 19
  - 4.4.1 Some definitions . . . 19
  - 4.4.2 Mapping devices onto memory map . . . 19
  - 4.4.3 Binary address mapping . . . 21
  - 4.4.4 Partial decoding . . . 22
- 4.5 External memories and I/O ports . . . 23
  - 4.5.1 Memory map of MC68HC11D0 . . . 23
  - 4.5.2 Memory allocation . . . 24
- 4.6 Debugging tools . . . 26
  - 4.6.1 Oscilloscope test loops . . . 26
  - 4.6.2 Logic analyzers . . . 26
- 4.7 Wire-wrapping and other practical details . . . 27

5 BUFFALO Monitor **28**
- 5.1 BUFFALO commands' list . . . 28
  - 5.1.1 Block fill -BF . . . 29
  - 5.1.2 Breakpoints -BR . . . 29
  - 5.1.3 CALL SUBROUTINE . . . 30
  - 5.1.4 GO . . . 30
  - 5.1.5 HELP . . . 30
  - 5.1.6 LOAD . . . 30
  - 5.1.7 MEMORY DISPLAY - MD . . . 31
  - 5.1.8 MEMORY MODIFY - MM . . . 31
  - 5.1.9 MOVE . . . 32
  - 5.1.10 PROCEED/CONTINUE -P . . . 32
  - 5.1.11 REGISTER MODIFY -RM . . . 33
  - 5.1.12 TRACE -T . . . 33
- 5.2 Interrupt vectors . . . 34
- 5.3 Utility subroutines . . . 35
- 5.4 S-record information . . . 36
- 5.5 S-record content . . . 36
  - 5.5.1 S-record types . . . 36

5.5.2 S-record creation 37

1 Building a microcontroller system

Let us start with an inventory of knowledge we posses and identify those topics which could be relevant to the the design[1] of the system. As an example, a course on *digital circuit design* would provide a solid foundation whereas a course on *computer organization/architecture* would provide a perspective from higher grounds, which in turn would facilitate navigation through numerous realization details. On the other hand, courses on *software design/programming* would provide valuable guidance in designing structured programs, especially when using (low-level) assembler languages.

1.1 Design

The recommended design path is

$$\text{Architecture} \rightarrow \text{Implementation} \rightarrow \text{Realization}$$

where

- *Architecture* defines the functional appearance of a system to its immediate user
- *Implementation*[2] provides the logic structure that gives shape to architecture (architecture defines *what* happens whilst implementation *why* it happens)
- *Realization* is a concrete version of an implementation; considering components to be used, their interconnections, positions, shielding, packaging, components' reliability, etc.

1.2 Project

The above design path, from specifications (architecture) to realization (building) of a microcontroller system, could be better tackled considering it a project. Let us call it simply 'Learning to build microcontroller systems'.
The aim of the project is *learning*,acquiring theoretical and practical knowledge in building microcomputer systems and the goal is the *realization* of a functional microcomputer system.
Throughout the project we would *learn by doing*[3], learn from achievements, errors, from other students, instructors, from reference manuals, books and other documentation.
A system doing what? would be the first legitimate question posed by system architecture. Well, we would design a MS with an *application* in mind, where application would be defined as a system with a microcontroller *embedded* in it—such systems are generally called *embedded systems*. In this case it is the embedded system that defines the overall functionality or *architecture* of the system. For pedagogical reasons we would work with two architectures in mind: that of (i) a simple evaluation board(EVB) and (ii) an application of our own choice based on the EVB built.
Where to start? Which tools are needed? are two questions addressed in two following sections.

[1] Design (*n*) 1. The act of working out the form of something (as by making a sketch or outline or plan), 2. An arrangement scheme, 3. Something intended as a guide for making something else, 4. An anticipated outcome that is intended or that guides your planned actions, 5. A preliminary sketch indicating the plan for something, 6. The creation of something in the mind. WordWeb ©2001 by Princeton University

[2] which provides practical means for accomplishing something

[3] Experience, like a candle, illuminates the one who bears it.

TASK 1.1
Compare (a) a sand hour glass, (b) a wrist mechanical watch, (c) a quartz watch and (d) the Big Ben[4] from the architecture, implementation and realization points of view. What do they have in common?

1.3 The MC68HC11 microcontroller family

A good start is getting familiar with the 68HC11 microcontroller family, especially with the the MC68HC11D0 , the low-end member of the family we would built the system with. Figure 1 shows a simplified block diagram of the 68HC11D0. The

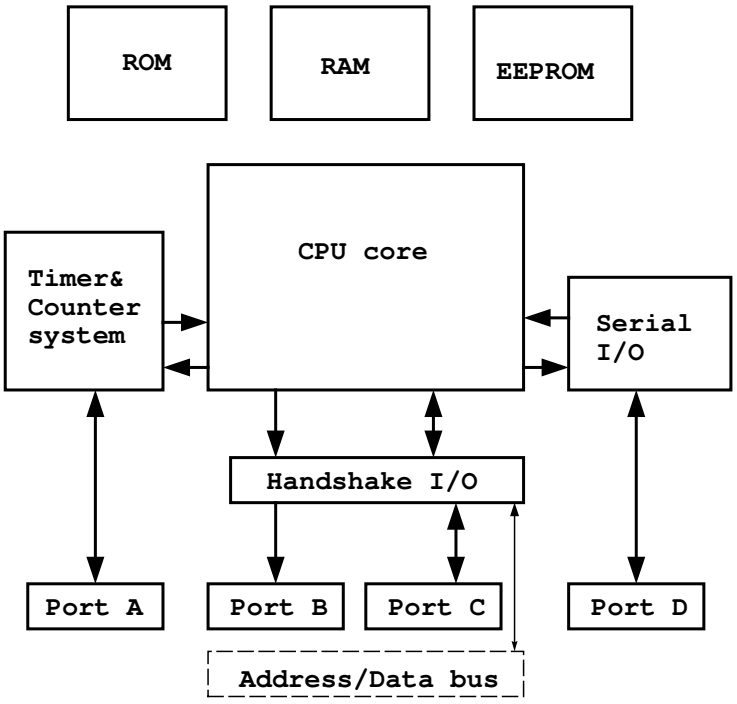


Figure 1: Simplified Block Diagram of a MC68HC11D0

block diagram reveals the basic features of the microcontroller. The *central processing unit*(CPU) core[5] is shown at the center. The microcontroller has three *internal* memories, respectively ROM, RAM and EEPROM (the acronyms stand respectively for 'Read Only Memory', 'Random Access Memory' and 'Electrically Erasable Programmable Read Only Memory').

[4]Housed inside The Houses of Parliament in London the clock has four dials of 23 feet square, the minute hand is 14 feet long and the figures are 2 feet high. Minutely regulated with a stack of coins placed on the huge pendulum, Big Ben is an excellent timekeeper, which has rarely stopped.

[5]CPU is the essential part of the chip, hence the name CPU core

The *timer/counter* unit/(sub)system, shown on the left of CPU, is a versatile timing unit with many many functions, such as generation of pulses and rectangular waveforms, measurement of periods, pulse widths, etc. The timer unit can be quite useful in *real-time* system application, where timing of events is crucial to the system. The serial input/output (I/O) systems provides two serial units: the *serial communication interface* (SCI) and the *serial peripheral interface*(SPI). The former is a standardized EIA232 interface (known also as RS232) and it is used for communications between various remote devices (above several meters). The SPI is an synchronous interface used often for communication between local subsystems, e.g. ICs in a printed circuit board (PCB) or devices inside a car).
The MC68HC11D0 has four ports A,B,C, and D. They can be configured to be general purpose inputs, outputs or bidirectional ports. Ports A and D are associated with timer/counter and serial I/O units. Ports B and C provide for advanced I/O, where data exchange is synchronized by *handshake signals*.
The discussion so far is focused on the *single chip mode* or configuration, where the whole system is contained in the 68MC11D0 chip. In cases where more resources are needed, such as extra RAM, PROM, I/O ports, the processor can be configured in *expanded mode*. In this mode ports B and C are not any longer available, instead, the pins are associated with a *multiplexed address/data bus*. Two interface signals would serve as *address strobe* (AS) and *read/write* (R/$\overline{\text{W}}$) signals for the control bus.

1.4 Development tools

Typical tools in system development are *evaluation boards* (EVB) and *development systems*(DVS). The firsts are simple systems build upon the same microprocessor/ microcontroller as the target system (the system we intend to build). Generally the EVB, besides the processor, have RAM and ROM or different kind of PROM(programmable ROM), *input* and *output* ports, *serial communication interface*(SCI), and a *monitor* program that resides in (ROM/PROM) and provides *operating environment*.
The monitor maintains the communication with a terminal (nowadays a *terminal program* on a general purpose computer), provides a set of utilities and various subroutines the developer can use. EVB, as the names implies, are used to evaluate the performance of a given processor. Often they do provide tools for program debugging (finding errors), tracing executions of the instructions, breaking the program flow and showing the content of the registers, reading/modifying memory content at different locations, etc. Section 5 describes the "BUFFALO" monitor we would work with when developing our MCU system.

2 Implementation of a complete single-chip system

A complete single-chip system (for the MC MC69HC11A8) is depicted in Figure 3.8 on the MOTOROLA's[6] reference manual, referred thereafter as *the manual*.

[6]©MOTOROLA INC.

Insert here the Figure ’Basic Single-Chip Mode Connections’ from the reference manual.

Referring to MC68HC11D0 we observe that it has no port E (analog to numerical converter). Removing the pins associated with this port,as well as reference pins VRH and VRL and components associated to it, we obtain the schematics of a 68HC11D0 microcontroller system in single ship mode.
We need observe that all the inputs pins are connected (pulled up) to 5V through 'pull-up' resistors; typical values are 10 kΩ. The 5V power supply is connected to pins VDD and VSS[7]. An eight MHz quartz is connected to pins EXTAL and XTAL according to the figure. The last component is a undervoltage sensor which generates a *reset* signal when power supply is turned on (*power-on-reset* circuit)—read the manual and circuit's data sheets for more information.
If we would connect the MC68HCD0 as shown in the reference manual, would the system work? How would we know if this was case? If it works, what is it it doing?
Assembling[8]a hardware around a microcontroller does not make a functional system. A microprocessor system[9], is supposed to perform its tasks by running the necessary software, which at the last stage, should be the application software.
During the development phase we would probably need another sort of software which should support system development. As we discussed previously, EVB and DVS are tools used for that purpose.
The MC68HC711D0 is delivered with 4kB ROM, which often contain a monitor program. When the processor is in single chip mode, a RESET signal forces the controller to execute the monitor program, see Section 5. Amongst others, the monitor program initialize various units such as the SCI and wait for serial inputs from outside world. To sent serial information to SCI we can practically use any terminal program available.
In general, the information to be sent to the processor should be 'commands' that the monitor understands and data it needs. As an example sending the command 'load' to the monitor set the monitor to wait for program code. In receiving the program code the monitor writes it in required addresses of available memory and then executes the program from its start address.
In order to sent program code to the monitor, the program source file should be assembled or compiled (by other programs such as assemblers or compilers) to program code (hex code) the processor understand. A DVS could integrate the above jobs in a single user-friendly environment[10].
So far, supposing that we have learned the functionality of ETERM, we would be able to write a program in assembler, assemble it, (down)load and execute it in the *target processor*. If the program is not working, or (even worse) working unsatisfactory, we would need to debug[11]the program using ETERM utilities such as *break*, *trace*, *registers*, etc. A complete list and description of BUFFALO monitors commands is given in Section 5.
.

QUESTIONS 2.1

[7]the symbols refer to *drain* and *source* pins of CMOS (complementary metal oxide) transistors, the technology MC68HCD0 is build upon

[9]in difference from nonprogrammable digital circuits

[10]ETERM© by GMV is such a system we would use during the course

[11]On November 18, 1878, Edison wrote in a letter to a European representative: "It has been just so in all my inventions. The first step is an intuition and it comes with a burst, then difficulties arise – this thing gives out and then that–"*Bugs*"–as such little faults and difficulties are called – show themselves, and months of anxious watching, study and labor are requisite before commercial success–or failure–is certainly reached" (Matthew Josephson, Edison: A Biography, John Wiley & Sons, 1992, page 198)

1. *What are the pull-up resistors used for?*

2. *Why a quartz is connected between pins EXTAL and XTAL. What is the purpose of the surrounding components?*

3. *What are the functions of pins MODA and MODB.*

4. *What are the pins XIRQ and IRQ ? Are the pull-up resistors necessary for these pins? Could the pins be connected to the ground?*

5. *Why is the resident program called monitor?*

6. *Given a functional system in single-chip mode, look at microcontroller's pin E with an oscilloscope. (i) What is the function of pin E. (ii) Measure the frequency of the output waveform.*

7. *What is the purpose of decoupling capacitors.*

2.1 Interfacing SCI with a terminal

As mentioned in section 1.3, MC68HC11D0 provides a SCI subsystem. Figure 2 depicts conversion of SCI's TTL level signals to $\pm$10V signal levels required from the EIA232 standard.
Some specification of the the MAX232[12]IC are given in the Figure 2. Connecting few external capacitors to it, according to IC's data sheet Figure 3, we could provide a RS232 interface which could allow to connect the MCS to a terminal, e.g. ETERM.
,

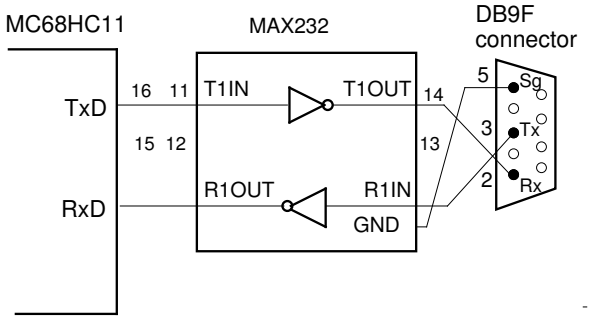


Figure 2: Serial connection of the MC8HC11 to a terminal serial port

Note the crossed connections T1OUT–Rx and R1IN–Tx, referred to as *null modem*.

[12]The MAX220-MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where 12V is not available

PIN	PURPOSE	SIGNAL NAME
Pin 1	Data Carrier	Detect DCD
Pin 2	*Received Data*	RxData
Pin 3	*Transmitted Data*	TxData
Pin 4	Data Terminal Ready	DTR
Pin 5	*Signal Ground*	Gnd
Pin 6	Data Set Ready	DSR
Pin 7	Request To Send	RTS
Pin 8	Clear To Send	CTS
Pin 9	Ring Indicator	RI

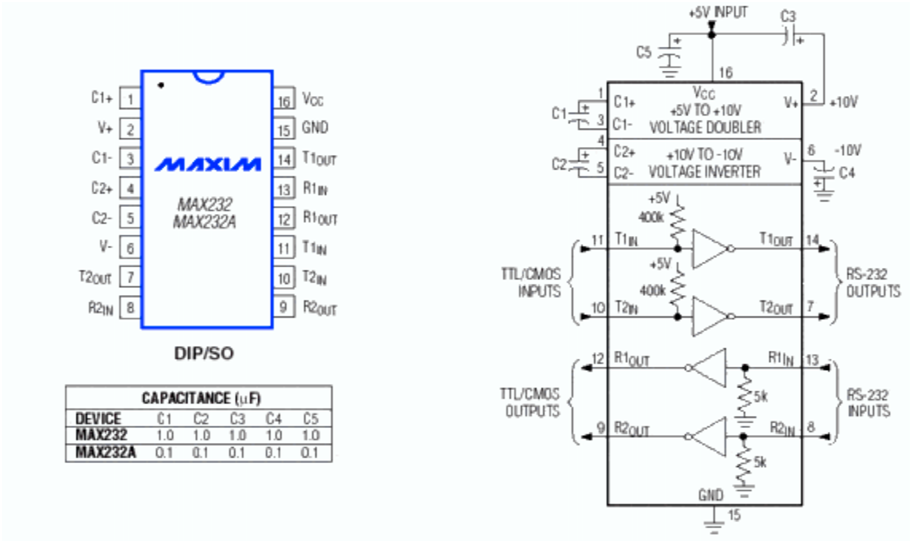


Figure 3: MAXIM 232A

Configuring the ETERM communications parameters, such as *Baud rate*, to the default BUFFALO monitor settings (9600 Baud), we would be able to establish a connection between the two systems. By resetting the system in single-chip mode we would be able to observe the BUFFALO prompt on ETERM's terminal. The prompt indicates that we have succeeded to built a complete MCS! The MC8HC11D0 has 192 bytes static RAM and 4kB ROM with a simple version of BUFFALO monitor in it.
We can now develop our own programs in RAM [13] or, depending on the family member, program them in PROM/EEPROM.

PROBLEM 2.1

1. Using a DIP switch and few resistors show how to set the microcontroller to four different modes.
2. Draw the RS232 signal of letters 'A' and 'a'

[13] notice that the RAM is volatile and its contents 'flies away' when power is turned off

3. Display the SCI signal after $\overline{\text{RESET}}$. What is the BAUD rate? Amplitude?

4. Check with oscilloscope the signal +5V connected to the processor. What do you observe?

3 Debugging

What if do not observe the expected BUFFALO prompt?
Well, hum..., something is definitely wrong. Anyhow, there is no cause for alarm. "Errare humane est." A series of questions arise naturally: Why? Why us? What could have gone awry? How?
The debugging process is often quite time-consuming and demands considerable mental efforts and skills. In the following section we would discuss a debugging method. Debugging/developing tools will be discussed on section3.1.

3.1 Debugging methods

Debugging is a part of system design. If at this time we have good documentation at hands, such as schematics, data sheets of ICs used, etc., we would probably solve the problem(s) sooner then later and get a useful experience on the way.
The strategy "divide et impera" (divide and conquer) could be used. The system is divided in parts, the parts are checked for bugs and if a bug is found in a part this part is subdivided on smaller parts and so on. We emphasize that an accurate *documentation* on each design step and of implementation details is of a paramount importance on the debugging process. Let us follow a hypothetical debugging scenario of a carelessly designed[14] system. A set of questions is as tabulated below:

Questions	Assessed part	Complementary part	Answer
Is the system working[a]			no
Is the processor working[b]	processor	the rest	no
Is the processor hot?	processor	power supply	no
Is the processor on single chip mode?	processor	settings	no
action:reconnect properly			
Is the system working?			no
Is the SCI working?	processor	serial interface	yes
Is the MAX232 circuit working[c]?	MAX232	ETERM/Settings	no
...	...	...	...

[a] A buffalo prompt should appear on terminal; we expect to see an activity on pin16(TxD) of the processor as a serially formatted rectangular signal of the ASCII characters "Buffalo...".)

[b] The E-Clock and SCI output should be observed on oscilloscope. Revisit also the answers of questions 2–4 and 7 in the section 2.1

[c] A $\pm$10V RS232 signal should be observed

Proceeding as above we would soon have a full functional system and along with it an useful experience on building, testing, debugging (on-chip) MC68HC11 microcontroller systems.

[14] not ours!

[14] The book 'The Adventures of Sherlock Holmes', published at 1891 by Conan Doyle, could be quite an inspiring complementary reading on 'debugging'—or at least a nice pastime.

4 A MCS in expanded mode

The design strategy of microcontroller chips is to minimize the number of external components by providing specific functionalities. Nevertheless, there are applications that require more resources that are provided by the chip. In such cases the microcontroller can provide an external address and data bus in expense to specific functionalities (I/O ports).
Referring to block diagram inFigure 1 we observe that in expanded mode we loose ports B an C along with their associated handshake signals. Instead, we obtain a multiplexed address data bus along with control lines R/$\overline{W}$ (read/write) and AS(address strobe).
Figure 4 show a generic bus system connected to various devices.

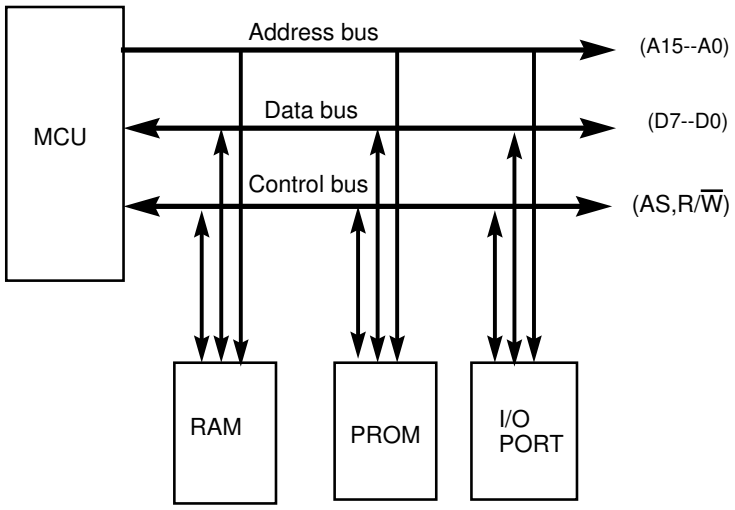


Figure 4: Generic bus system and its interface with various devices

4.1 Bus demultiplexing

Many of the members of the MC68HC11 family have multiplexed address/data bus. In Figure 5 the address/data lines AD7–AD0 are used for both addresses and data; first the processor issues addresses and then data. The process is called time multiplexing. The address strobe (AS) signal turns active high when there is a valid address in the address/data bus. The signal can be used to latch the address (e.g. with a 74HC373 latch). The control bus is unidirectional, with output signals R/$\overline{W}$, AS, and E.

4.2 Timing diagrams

Figure 6 shows a *write cycle* time diagram of MC68HC11D0 in expanded mode. [15] The negative flank of the AS signal latches the addresses A7 to A0 as shown in Figure 5. At this instant the all addresses A15–A0 are available.

[15] The diagrams are schematically drawn; the reader should refer to the manual for complete and accurate time diagrams.

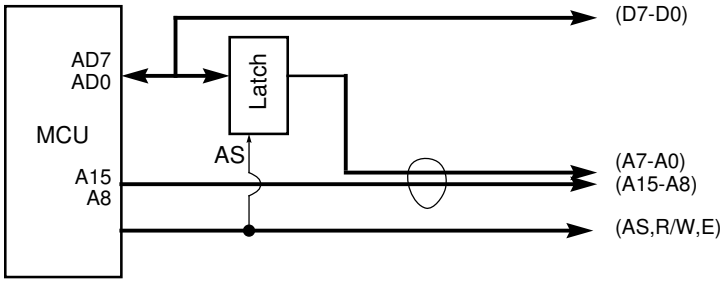


Figure 5: Demultiplexing of multiplexed address/data bus of MC68HC11

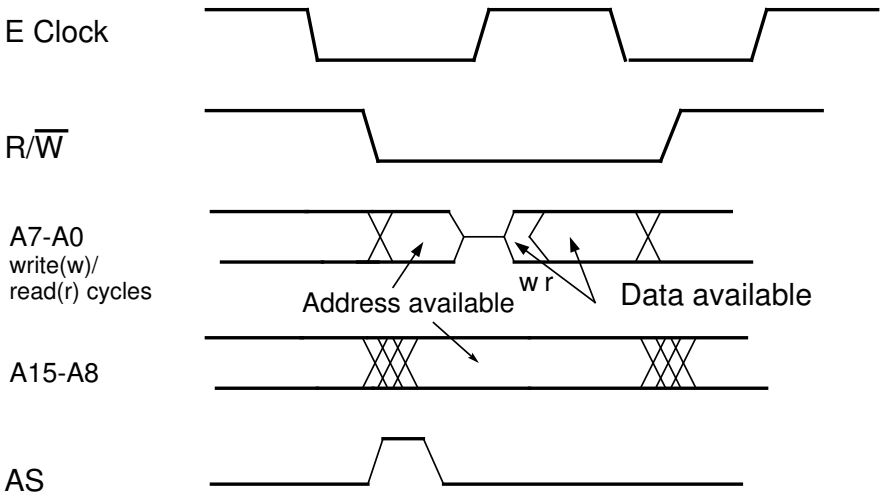


Figure 6: Time diagram of read/write cycles (expanded mode)

The signal R/$\overline{\text{W}}$ is low on a write cycle, and the data would be available in the data bus from time instant marked by 'w'. In case of a read cycle the signal R$\overline{\text{W}}$ is high and the data would be available from instant 'r' in the Figure 6.
A useful observation is that the data are available on the second cycle of E clock, when E clock signal is high. As the time interval of R/$\overline{\text{W}}$ signal covers both address and data signals the E clock should be included in the control bus, to ensure that we write or read data and not address information instead of it.
Figure 7 shows how the E signal is decoded to generate read and write signals[16] for some typical memory ICs.
When the E signal is low both $\overline{\text{WE}}$ and $\overline{\text{OE}}$ signals become high, independently of the input R/$\overline{\text{W}}$ signal. When the E signal is high then $\overline{\text{OE}}$ = R/$\overline{\text{W}}$ = $\overline{\text{R}}$/W and $\overline{\text{WE}}$ = R/$\overline{\text{W}}$ ($R \rightarrow \overline{\text{OE}}$ and $\overline{\text{W}} \rightarrow \overline{\text{WE}}$). The following figures show the connection of a system in a expanded mode.

[16]Three of four NAND gates of IC 74HCT00N could be used

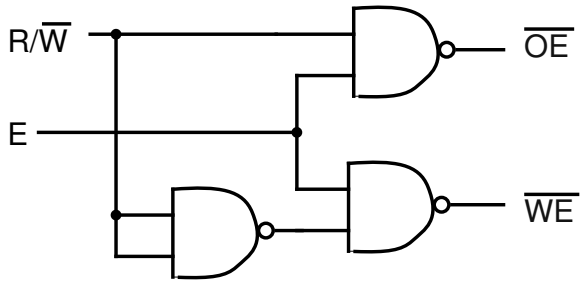


Figure 7: $\overline{WE}$ and $\overline{OE}$ (or $\overline{RD}$) signals used to interface with some typical memories

4.3 Tristate bus

The processor bus provides the highway of communication of the MCU with various devices such as memories, I/O ports, and other special peripheral devices.

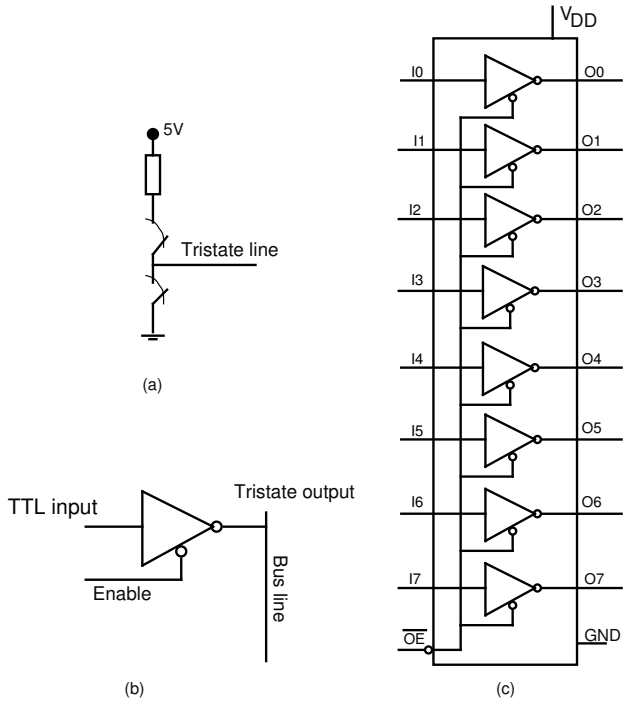


Figure 8: Tristate lines and buffers

Two devices should not be allowed to simultaneously sent signals on the same data line. If an high and low level signal are sent simultaneously the signal level in the line will be undetermined and device damage might occur.

FAIRCHILD SEMICONDUCTOR™

September 1983
Revised February 1999

MM74HC373
3-STATE Octal D-Type Latch

General Description

The MM74HC373 high speed octal D-type latches utilize advanced silicon-gate CMOS technology. They possess the high noise immunity and low power consumption of standard CMOS integrated circuits, as well as the ability to drive 15 LS-TTL loads. Due to the large output drive capability and the 3-STATE feature, these devices are ideally suited for interfacing with bus lines in a bus organized system.

When the LATCH ENABLE input is HIGH, the Q outputs will follow the D inputs. When the LATCH ENABLE goes LOW, data at the D inputs will be retained at the outputs until LATCH ENABLE returns HIGH again. When a high logic level is applied to the OUTPUT CONTROL input, all outputs go to a high impedance state, regardless of what signals are present at the other inputs and the state of the storage elements.

The 74HC logic family is speed, function, and pin-out compatible with the standard 74LS logic family. All inputs are protected from damage due to static discharge by internal diode clamps to V_{CC} and ground.

Features

- Typical propagation delay: 18 ns
- Wide operating voltage range: 2 to 6 volts
- Low input current: 1 μA maximum
- Low quiescent current: 80 μA maximum (74 Series)
- Output drive capability: 15 LS-TTL loads

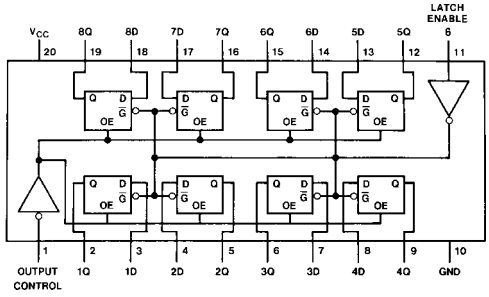
Ordering Code:

Order Number	Package Number	Package Description
MM74HC373WM	M20B	20-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-013, 0.300" Wide
MM74HC373SJ	M20D	20-Lead Small Outline Package (SOP), EIAJ TYPE II, 5.3mm Wide
MM74HC373MTC	MTC20	20-Lead Thin Shrink Small Outline Package (TSSOP), JEDEC MO-153, 4.4mm Wide
MM74HC373N	N20A	20-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300" Wide

Devices also available in Tape and Reel. Specify by appending the suffix letter "X" to the ordering code.

Connection Diagram

Pin Assignments for DIP, SOIC, SOP and TSSOP



Top View

Truth Table

Output Control	Latch Enable	Data	373 Output
L	H	H	H
L	H	L	L
L	L	X	Q_0
H	X	X	Z

H = HIGH Level
L = LOW Level
Q_0 = Level of output before steady-state input conditions were established.
Z = High Impedance

© 1999 Fairchild Semiconductor Corporation

[17] bus - 1832, abbreviation of omnibus (q.v.), from Fr. 1820s voiture omnibus "carriage for everyone." The Eng. word is simply a Latin dative plural ending.

Insert here ’Basic Expanded Mode Connections (Sheet of 1 of 2)’ from the reference manual

Insert here 'Basic Expanded Mode Connections (Sheet of 2 of 2)' from the reference manual

4.4 Address Decoding

4.4.1 Some definitions

To individually *address* various devices we would need to give them specific *addresses*. The available *address space* size S is dictated by the number of address bus lines (bus width) N, namely $S = 2^N$. In case of 16 bits bus $S = 2^1$. The the address space spans the region \$0000–\$FFFF [18] or 0–65535 in decimal format. Figure 9 shows the address space of an 16-bits address bus. Start and end addresses of 8kB blocks are shown respectively on the left and right of address space.
Each device is allocated (maps to) a specific region on the address space. A diagram that displays the allocated regions or *partitions* on the address space is called *memory map*. As an example, Figure 10 shows memory map of a MC68HCA1/A8 MCS. Shaded surfaces show memory regions that are occupied by internal (on-chip) memories. In expanded mode, external memories can be mapped onto remaining address space (unshaded surfaces). In single-chip mode these regions are not available.

4.4.2 Mapping devices onto memory map

We introduce the subject by an example.

EXAMPLE 4.1
Assume that a MC system has two memory ICs of 32KB each. Memory chips of these size have 15 address pins ($2^{15} = 32$K) A14–A0, eight data pins D7–D0, and few control pins, such as $\overline{\text{OE}}$, $\overline{\text{WE}}$, and $\overline{\text{CS}}$[19]. Bus address lines A14–A0 should be connected to the pins A14–A0 of the memory ICs. The address line A15 could then be used to generate chip select ($\overline{\text{CS}}$) signals for the two memories as shown in Figure 11, i.e.,

$$\overline{\text{CS}}_1 = \text{A15 and } \overline{\text{CS}}_2 = \overline{\text{A15}}.$$

When a $\overline{\text{CS}}$ is low memory's data pins connects to the data bus, otherwise they are disconnected. When A15 is zero, $\overline{\text{CS}}_1$ signal selects the RAM (\$0000-\$7FFF) whereas when A15 one, $\overline{\text{CS}}_2$ signal selects the EEPROM (\$7FFF-\$FFFF).

[18] The symbol '\$' indicates that the number is given in hexadecimal format
[19] PROMs might have additional control pins for programming

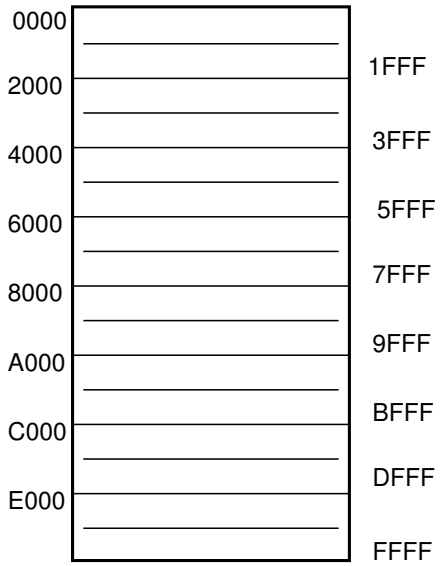


Figure 9: Address space of a 16 bit address bus

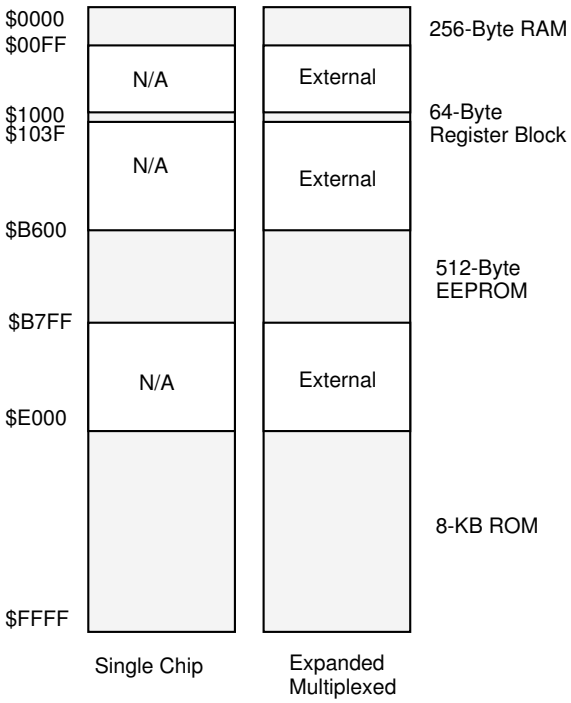


Figure 10: Memory map of MC68HCA1/A8

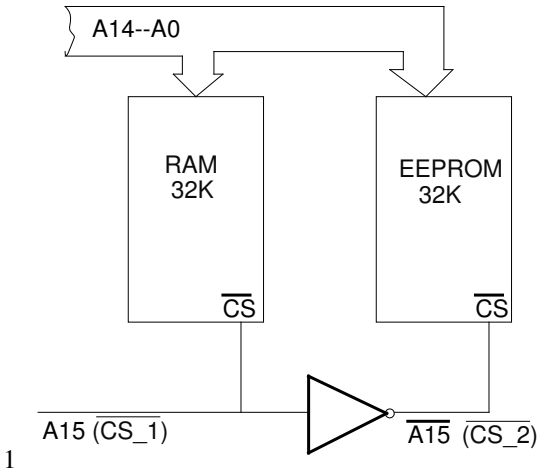


Figure 11: Chip select($\overline{\text{CS}}$) signals of two memory chips

In general we need to construct a digital circuit (decoder) that monitors the address bus and generates chip select signals to memory chips according to memory map. The process is called *address decoding*.
The same decoding scheme of Figure 11 could be applied to two smaller sizes memories. This case, referred to as *partial decoding*, does not use the whole available address space. Some of upper addresses lines would be left unconnected. We would revisit this topic on the following sections.

4.4.3 Binary address mapping

If the MCS on the example 4.1 had a number of 4kB (of 2^{12} bytes) memory chips then, to assess their content, we would need 12 address lines (A11–A0). The rest four of address lines A15–A12 could be used for address decoding. in this case we obtain sixteen (2^4) partitions where sixteen memory chips could be individually selected.
We notice that the product of memories of sizes 2^M with the number of memory chips 2^{N-M} is a constant 2^N ($2^M \times 2^{N-M} = 2^N$. In case of a 16 bit address bus $2^{12} \times 2^4 = 2^{16}$. It can be seen that the 64K address space of a 16 bits address bus can accommodate *at most* 2^N memory ICs of sizes 2^{16-N}, where N = 0, 1, ... 15.
If we had a 64kB EEPROM (2^{16}) and as consequence a single ICs (2^0), then the 16 address lines should be connected to 16 address pins A15–A0 of the memory. The $\overline{\text{CS}}$ signal of the EEPROM should then be grounded.

QUESTIONS 4.1

1. *Design a decoding scheme for addressing 2^{16} memories of one byte each. How does it compare to the example of decoding a single 64kB memory. Do the extremes meet?*

2. *What is the maximum number of 32K memory chips that can be accommodated on the memory space of a 32bit address bus?*

e

4.4.4 Partial decoding

Assume that an MCS needs 4KB RAM, 16KB EEPROM and two I/O ports respectively with three and eight registers. We could make a binary partition as in Section 4.4 of the address space to 2, 4, 8, and or more regions.
Let us locate the RAM on the memory map at addresses $8000–$8FFF. Note that $8000 + $1000 -$0001 =$8FFF, where $1000 = 4096. Only in this this address interval we should *enable* the external RAM chip.
Let's display the address range in the binary format as shown in the following table. symbol x denotes an address line which can be either 0 or 1.

Address	A15	A14	A13	A12	A11	A10	...	A0
$8000	1	0	0	0	0	0	...	0
$8001	1	0	0	0	0	0	...	1
$8xxx	1	0	0	0	x	x	...	x
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$8FFE	1	0	0	0	1	1	...	0
$8FFF	1	0	0	0	1	1	...	1

From this table (consider particularly the third row), we can generate the *chip enable signal*($\overline{\text{CS}}$) by the Boolean expression:

$$\overline{\text{CS}} = \overline{\text{A15}\ \ \overline{\text{A14}}\ \ \overline{\text{A13}}\ \ \overline{\text{A12}}}$$

Let us conclude thus section with two more examples.

EXAMPLE 4.2
Map an I/O port with three registers from address $4000. Design the address decoding circuit. The address range is $4000–$4002 is defined by address range 010000000000000xx where xx can takes values 00,01,10,11. To generate a $\overline{\text{CS}}$ signal for (only) the above locations would required to decode 14 address lines, from A15 to A2.

$$\overline{\text{CS}} = \overline{\overline{\text{A15}}\ \ \text{A14}\ \ \overline{\text{A13}}\ \ \overline{\text{A12}}\ \ \overline{\text{A11}}\ \ \ldots\overline{\text{A2}}}$$

Note that the scheme includes also the address $4003 although no register is located there. As expected, the decoding circuit gets quite large.

As mentioned earlier, when we need not consume all the available *address space* we can make a partial decoding, i.e. allocate a larger address range to the I/O port, or other devices, although they do not need or use it. The address decoding circuit simplifies considerably.

EXAMPLE 4.3
Assigning the range $4000–$43FF to the I/O port above we can generate $\overline{\text{CS}}$ signals by Boolean expression

$$\overline{\text{CS}} = \overline{\overline{\text{A15}}\ \text{A14}\ \ \overline{\text{A13}}\ \ \overline{\text{A12}}\ \ \overline{\text{A11}}\ \ \overline{\text{A10}}}$$

Note that all addresses with binary patterns 010000xx...x, as tabulated below, will generate a single $\overline{\text{CS}}$ signal, i.e select the same I/O port. The lower two bits of address bus determine which registers to be accessed.

$4000–$4003, $4004–$4007, $4008–$400B, $400C–$400F,
$4010–$4013, $4014–$4017, $4018–$401B, $401C–$401F,
$4020–$4023, $4024–$4027, $4028–$402B, $402C–$402F,
...
$4100–$4103, $4104–$4107, $4108–$410B, $410C–$410F,
$4110–$4113, $4114–$4117, $4118–$411B, $411C–$411F,
...
$4200–$4104, $4104–$4107, $4108–$410B, $410C–$410F,
...
$42F0–$42F3, $42F4–$42F7, $42F8–$42FB, $42FC–$42FF,
...
$43F0–$43F3, $43F4–$43F7, $43F8–$43FB, $43FC–$43FF

The addresses $4000, $4001–$4002 (and $4003) can be used to assess registers. The remaining I/O space contains replica of these registers and can not be used by other devices.
The following VHDL could be used to program a PLD (programmable logic device). e.g. a PAL device (programmable array logic) for address decoding.

```
library ieee;
use ieee.std_logic_1164.all;
entity address_decoder is
    port( A15,A14,A13,A12,A11,A10 :in std_logic;
          CS_L: out std_logic);
end address_decoder

architecture decoder_arch  of  address_decoder is
begin
    CS_L <= A15 and not(A14) and A13 and A12 and A11 and A11;
end decoder_arch;
```

QUESTIONS 4.2

1. *What is the rationale of using address lines for generating $\overline{CS}$ signals?*
2. *Show that positioning a 16kB memory from an address like $3B59 would require a more complex decoding circuit then positioning it at an address $4000.*
3. *A MC system should have three external devices connected to the bus: RAM 8KB, EEPROM 16KB, ROM 2KB. In addition a 4kB address space should be reserved from future expansion. Design a minimal address decoding circuit.*
4. *Design the decoding circuits of two 16kB EEPROM memories. Referring to Figure 11, how many address lines would be connected to memory ICs? What happens with the rest of address lines?*

4.5 External memories and I/O ports

In this section we discuss the design of a comprehensive MCS based on MC68HC11D0 microcontroller set expanded mode. In previous section we have introduced ample information on design of a microcontroller based system with reference to MC68HC11 family of microcontrollers.

4.5.1 Memory map of MC68HC11D0

On the following we discuss memory mapping of external devices to the address space of MC68HC11D0 . For that the microcontroller should be set in expanded mode[20]. ,

[20]Special bootstrap mode will be discussed later on whereas special test mode is not considered

by setting proper values to MODA, MODB pins of the processor.
To expand the system with external devices we need first refer to memory map of MC68HC11D0 , especially in expanded mode. Figure 12 shows its memory map.

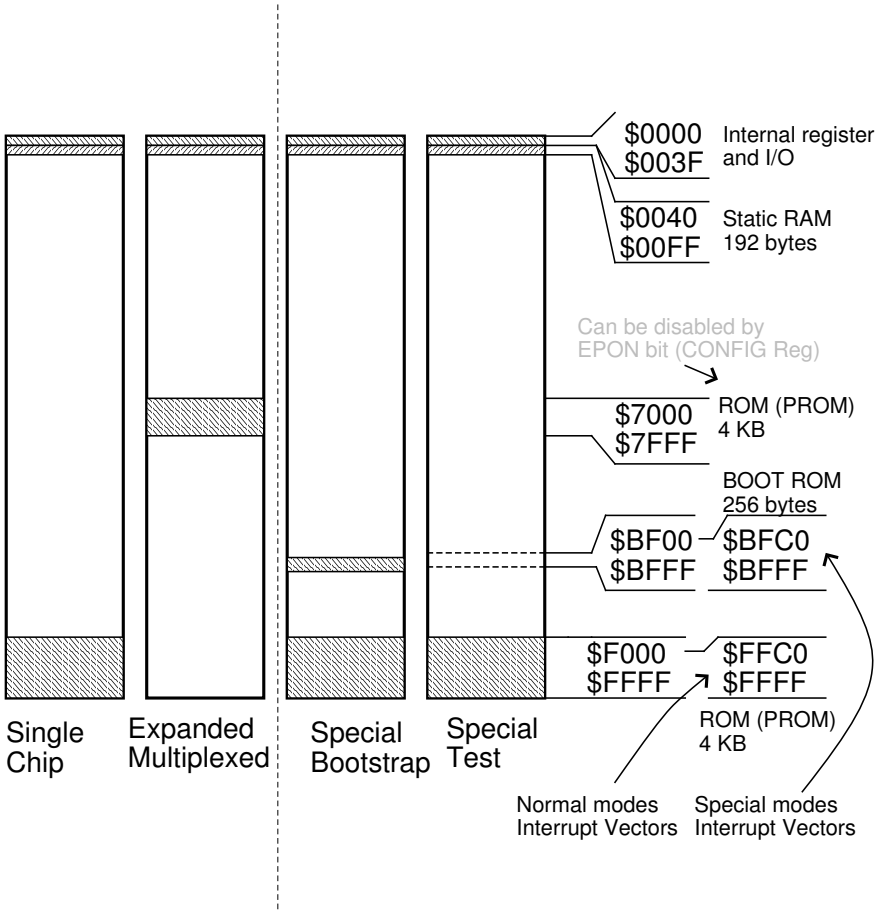


Figure 12: Memory map of MC68HC11D0

At the top of memory map are allocated (in all modes) 64 register followed by 192 bytes of RAM. In single-chip mode 4K of PROM(ROM) are located on bottom of the memory map[21]. In expanded mode there is a 4kB PROM(EEPROM) which can be disabled shortly after the reset. Assuming it be the case, the all the address space, expect for top 256 bytes, is available to external devices.

4.5.2 Memory allocation

Considering that binary address decoding produces a simpler circuitry, we can partition the available address space in four 16kB regions. Three memories of 16kB each can be easily accommodated at addresses $4000, $8000, and $C000. Only two address lines

[21] as mentioned in section 2.1, a reduced version of BUFFALO monitor resides there

would need to be decoded to generate proper $\overline{\text{CS}}$ signals: $\overline{\text{CS}}_2$, $\overline{\text{CS}}_3$, $\overline{\text{CS}}_4$, respectively for three lower (shaded) regions inFigure 13. The region $2000-$3FFF could be saved for future system development.

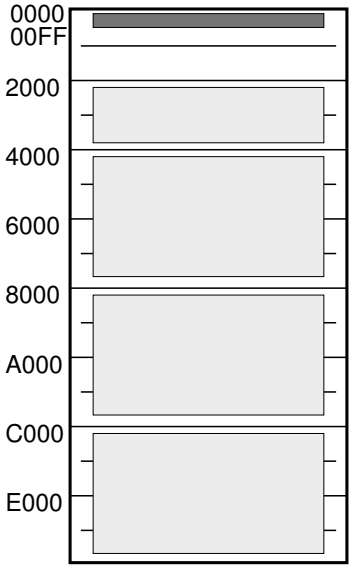


Figure 13: An arrangement of various devices on 68HC11D0 memory space

The shaded boxes in Figure 13 depict allocated memory space (devices there could fill the whole allocated region. The darker box on top of the memory map depicts the internal register and RAM memory space.
We could allocate the region $C000–$FFFF to a FLASH memory[22]. Consider that normal mode RESET and interrupt vectors should be located there, see Figure 12. The RESET vector occupies the two last bytes.
RAM could be located e.g. from address $8000. I/O ports, which in general contains few registers, are considered as memory (the method is referred to as memory mapped I/O). Input and output ports can be mapped respectively in the two remaining regions $2000 and $4000. The input port can be selected by $\overline{\text{CS}}_{1,2} = \overline{\text{CS}}_1 + \overline{\text{A13}}$, where symbol '+' denotes the logical OR operation, $\overline{\text{CS}}_1$ selects the first region ($0000–$3FFF) and $\overline{\text{CS}}_{1,2}$ selects the lower part of it[23].
Another solution could be to split the region $4000–$7FFF in two parts, respectively for input and output ports, and save the region $2000–$3FFF for future development needs.

PROBLEM 4.1
Design the address decoding circuit for mapping an input and an output port at address ranges $4000–$5FFF and $6000-$7FFF

[22]FLASH memory technology offers few advantages over PROM; such as a single 5V power supply, which is advantageous in firmware field updates.

[23]the region $1000–$1FFF (or even $0100–$1FFF) could also be used with a more complex decoding circuity

4.6 Debugging tools

Here we consider two debugging tools oscilloscope test loops and & logic analyzers.

4.6.1 Oscilloscope test loops

We mentioned earlier the oscilloscope as a measurement device. With it we can observe and measure parameters of time varying signals—in comparison, a multimeter is quite limited as it measures only DC values (or RMS value of 50Hz alternate signal). The oscilloscope is mostly useful in displaying periodic signals, such as the E-clock signal of MC68HC11 processor, or repetitive (periodic) signal events. One of the signal can be selected as a trigger, meaning that when its amplitude passes a chosen threshold the oscilloscope would start a new sweep. If the trigger signal was periodic we could observe a stable signal on the oscilloscope's display[24].
Let illustrate the oscilloscope use in hardware debugging by an example. Assume that we have build a MCS but (unfortunately) found out that we could neither read nor write in one of its external RAM chips.
The signals connected to the RAM in question were address bus lines (their number depending on the RAM size), data bus lines, signals $\overline{\text{CS}}$, $\overline{\text{OE}}$ and $\overline{\text{WE}}$. The last two signals were obtained according to Figure 7.
A source assembler code (for MC86HC11) for a small test loop is shown below

```
* Test loop of a memory chip. The program
* writes/reads cyclicly to/from a memory address
TEST       LDAA    #$AA
S_LOOP     STAA    MemAddress
*          LDAB    MemAddress
           BRA     S_LOOP
* End of program
```

The program stores data $AA (%10101010) to (an arbitrary) location MemAddres of the RAM—code line 3 is commented out but can be included to test the read cycle; code line 2 might be commented out in this case.
From the time diagram Figure 5 we observe that address information is put firstly on the address bus. Its decoding generates the $\overline{\text{CS}}$ signal. In a two channels oscilloscope, we can choose to measure the signals $\overline{\text{CS}}$ and $\overline{\text{OE}}$ or $\overline{\text{CS}}$ and $\overline{\text{WR}}$. Selecting as trigger source the $\overline{\text{CS}}$ signal, which comes before and ends after the $\overline{\text{OE}}$ and $\overline{\text{WE}}$ signal[25], we we expect to observe signals similar to the read cycles shown in Figure 5.
After assembling and loading the file to the target system (to another functional memory) we can run the test loop program. The loop makes the $\overline{\text{CS}}$ signal repetitive. Triggering the oscilloscope by $\overline{\text{CS}}$ signal, and adjusting oscilloscope's time scale we can observe the stable waveforms of $\overline{\text{CS}}$ and lowOE signals. If any of the signal is missing or in not as expected then we could debug further the source of the problem, e.g. into the scheme in Figure 7. With ongoing software loop we can measure other signals.

4.6.2 Logic analyzers

Logic analyzers are digital instruments of preference in monitoring computer bus signals. They are mainly used as development tools.

[24] In addition, digital oscilloscopes can be programmed to capture a single event signals.
[25] the later are valid only on the second (high) cycle of E clock; refer to Figure 5

Logic analyzers have many measurement probes (often in multiple of eights) and provide more versatile functions than oscilloscopes. In difference from the oscilloscope, which displays real-time signal waveforms, they display the logical levels (or the binary pattern) of captured 'bus activity' in a long sequences of many bus cycles. Triggering is done on a chosen word pattern instead of signal levels. The bus activity is sampled periodically (e.g. by E clock) and when the trigger pattern is met a period of bus activity is recorded and displayed. If the trigger pattern was chosen to be an instruction code then the bus activity from a specific place in the program could be traced.

4.7 Wire-wrapping and other practical details

Wirewrap is an technology to interconnect electronics that was popular few decades ago. It has the advantages that it is easily modifiable, and easy to create prototype systems[26] On the other hand it is quite labor intensive.
Considering the learning process we are going through and relatively small size of the system we are building we would take advantage of wire-wrapping technique, see [27]

[26]The wirewrap technology declined in use because the PC board technology costs dropped and allowed higher density. Also PC boards allowed for a more controlled signal environment which was required for faster logic see http://www.pdp8.net/wirewrap/wirewrap.html.

[27]http:// www.okindustries.com/products/4.1.1.1.htm for practical details

5 BUFFALO Monitor

The monitor BUFFALO program is the resident firmware for the evaluation boards(EVB), which provides a self contained *operating environment*. The monitor interacts with the user through predefined commands that are entered from a terminal. The user can use any of the commands supported by the monitor.
A standard input routine controls the EVB operation while the user types a *command line*. Command processing begins only after the command line has been terminated by pressing the keyboard carriage return (RETURN) key. The command line format is

$$> < \text{command} > [< \text{parameters} >](\text{RETURN})$$

where $>$ is the monitor *prompt*, <parameters> is an expression or address and (RETURN) is the return keyboard key. Symbols [ ] mean enclose optional fields and [ ]... enclose optional fields repeated. Fielsa are separated, comma or tab characters. All input number are interpreted as hexadecimal.
A command can be corrected using backspacing (CTRL-H) or aborting the command (CTRL-H or DELETE). After a command is entered pressing RETURN again will repeat the command. A maximum of 35 characters may be entered in a command line. The 36th character terminates the command and the message "Too long" is displayed.

5.1 BUFFALO commands' list

The monitor BUFFALO program commands are listed alphabetically by mnemonic in the table below. Each of the commands are described in detail following the tabular command listing.

BF <addr1> <addr2> <data>	Block fill memory with data
BR [-] [<address>]...	Breakpoint set
CALL [<address>]	Execute subroutinea
G [<address>]	Execute program
HELP	Display monitor commands
LOAD <T>	Download (S-records*) via terminal
MD [<addr1> <addr2>]	Dump memory to terminal
MM [<address>]	Memory modify
MOVE <addr1> <addr2> <,dest>	Move memory to new location
P	Proceed/continue from breakpoint
RM[p,y,x,a,b,c,s,]	Register modify
T [<n>]	Trace n ($01-$FF) instructions

Depending on BUFFALO version and EVB some additional commands to[28] are:

ASM[<address>]	assembler/dissambler
BULK	bulk erase EEPROM
BULKALL	bulk erase EEPROM + CONFIG register
LOAD <host download command>	dowload S-records via host port
TM	enter transparent mode
VERIFY <host download command>	compare memory to downloaded data via host port
VERIFY< T >	compare memory to downloaded data via terminal

[28] http://www.technologicalarts.com/myfiles/data/buffalo.pdf

5.1.1 Block fill -BF

BF <address1 > <address> <data>
where:
<address1> Lower limit for fill operation
<address> Upper limit fill operation
<data> Fill pattern hexadecimal value.

EXAMPLE	DESCRIPTION
>BF 0200 0230 FF	Fills each byte of memory in the range $0200–$0230 with data pattern $FF.
>BF 4000 4FFF 00	Fills each byte of memory in the range $4000 to $4FFF with data pattern $00.

5.1.2 Breakpoints -BR

The BR command sets the address into the breakpoint address table. During program execution, a halt occurs to the program execution immediately preceding the execution of any instruction address in the breakpoint table. A maximum of four breakpoints may be set. After setting the breakpoint, the current breakpoint addresses, if any, are displayed. Whenever the G, CALL, or P commands are invoked, the monitor program inserts breakpoints into the user code at the address specified in the breakpoint table.

BR	Display all current breakpoints.
BR <address>	Set breakpoint.
BR <addr1> <addr2>...	Set several breakpoints.
BR -	Remove all breakpoints.
BR -<addr1> <addr2>...	Remove <addr1> and add <addr2>.
BR <addr1> - <addr2>...	Add <addr1>, clear all entries, then add <addr2>.
BR <addr1> -<addr2>...	Add <addr1>, then remove addr2>.

>BR [-][<address>] [-] by itself removes (clears) all breakpoints; [-] proceed [<address.]... removes individual or multiple addresses from breakpoint table.
Breakpoints are accomplished by the placement of a software interrupt (SWI) at each address specified in the breakpoint address table. The SWI service routine saves and displays the internal machine state, then restores the original opcodes at the breakpoint location before returning control back to the monitor program.
SWI opcode cannot be executed or 'breakpointed' in user code because the monitor program uses the SWI (software interrupt) vector. Only RAM locations can be breakpointed. Branch on self instructions cannot be breakpointed.

EXAMPLE	DESCRIPTION
>BR 0203 0203 0000 0000 0000	Set breakpoint at address location 0203
>BR 0203 0205 0207 0209 0203 0205 0207 0209	Sets four breakpoints.
>BR 0203 0205 0207 0209	Display all current breakpoints.
>BR - 0209 0203 0205 0207 0000	Remove breakpoint at address location 0209.
>BR 0209 - 0209 0000 0000 0000	Clear breakpoint table and add BF09.
>BR - 0000 0000 0000 0000	Remove all breakpoints.

5.1.3 CALL SUBROUTINE

CALL [<address>] where: <address> is the starting address where user program subroutine execution begins.
The CALL command allows the user to execute a user program subroutine. Execution starts at the current program counter (PC) address location ,unless a starting address is specified. Two extra bytes are placed onto the stack before the return from interrupt (RTI) is issued so that the first unmatched return from subroutine (RTS) encountered will return control back to the monitor program. Thus any user program subroutine can be called and executed via the monitor program. Program execution continues until a breakpoint encountered, or the EVB reset switch S1 is activated (pressed).

5.1.4 GO

G [<address>]
where: <address> is the starting address where user program execution (free run in real time). The user may optionally specify a starting address where execution is to begin. Execution starts at the current program counter (PC) address location, unless a starting address is specified. Program execution continues until a breakpoint is encountered, or the EVB reset switch is pressed.

5.1.5 HELP

The HELP command enables the user available EVB command information to be displayed on the terminal CRT for quick reference purposes.

5.1.6 LOAD

LOAD S-RECORD, LOAD T
The LOAD command moves (downloads) object data in S-record format (see Appendix A) from an external host computer to EVB. As the EVB monitor processes only valid S-record data, it is possible for the monitor to hang up during a load operation. If an S-record starting address points to and invalid memory location, the invalid address message ”error addr XXXX” is displayed on the Terminal CRT (xxxx = invalid ad-

dress).

EXAMPLE	DESCRIPTION
> LOAD[29]T	command entered to download data from computer to EVB via host port.

5.1.7 MEMORY DISPLAY - MD

MD [<address1> [<address2>]] where: <address1> Memory starting address (optional).
[<address2>] Memory ending address (optional).
The MD command allows the user to display a block of user memory beginning at address1 and continuing to address2. If address2 is not entered, 9 lines of 16 bytes are displayed beginning at address1. If address1 is greater than address2, the display will default to the first address. If no addresses are specified, 9 lines of 16 bytes are displayed near the last memory location accessed.

EXAMPLE
>MD F7D0
F7D0 BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF .....
F7E0 BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF .....
F7F0 BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF .....
F800 BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF .....
F810 BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF .....
F820 BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF .....
F830 BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF .....
F840 BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF .....
F850 BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF .....
>MD 0200 0220
0200 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0210 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0220 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....

5.1.8 MEMORY MODIFY - MM

MM [<address>]
where: <address> is the memory location at which to start display/modify.
The MM command allows the user to examine/modify contents in user memory at specified locations in an interactive manner. Once entered, the MM command has several submodes of operation that allow modification and verification of data. The following sub-commands are recognized.

[29]ETERM provides a flexible interface to the LOAD command

KEY	DESCRIPTION
CTRL J or (Space Bar)	Examine/modify next location.
CTRL H or A	Examine/modify previous location.
/	Examine/modify same location.
RETURN	Terminate MM operation.

Following are some examples of the use of MM command.

EXAMPLE	DESCRIPTION
>MM 0700	Display memory location 0700.
0700 44 66/	Change data at 0700 and re-examine location.
0700 66 55A	Change data at 0700 and backup one location.
06FF FF BF (RETURN)	Change data at 06FF and terminate MM operation.
>MM 013C	Display memory location.
013C F7 C18E0 51 013C F7	Compute offset, result = $51.
>MM 0200	Examine location $0200.
0200 55 80 C2 00 CE C4	Examine next location(s) using (Space Bar).

5.1.9 MOVE

MOVE <address1> <address2> [<dest>]
where:
<address1> Memory starting address.
<address2> Memory ending address.
[<dest>] Destination starting address (optional).

The MOVE command allows the user to copy/move memory to new memory location. If the destination is not specified, the block of data residing from address1 to address2 will be moved up one byte. Using the MOVE command on EEPROM locations will program EPROM cells. The MOVE command is useful when programming EEPROM. As an example, a program is created in user RAM using the assemble, debugged using the monitor, and then programmed into EEPROM with the MOVE command. No messages will be displayed on the terminal CRT upon completion of the copy/move operation, only the prompt is displayed.

EXAMPLE	DESCRIPTION
>MOVE E000 E7FF 0200	Moves data from locations $E000-$E7FF to locations $0200-$09FF.

5.1.10 PROCEED/CONTINUE -P

>P
This command is used to proceed or continue program execution without having to remove assigned breakpoints. This command is used to bypass assigned breakpoints in a program executed by the G command.

5.1.11 REGISTER MODIFY -RM

RM [P,X,A,B,C,S]
The RM command is used to modify the contents of MCU program counter (P), index registers (X), (Y), accumulators (A), (B) and stack pointer (S).

EXAMPLE	DESCRIPTION
>RM	Display P register contents.
P-0200 Y-798 X-FF00 A-44 B-70 C-BF S-0054	
P-0207 0220	Modify P register contents.
>	
>RM X	Display X register contents.
P-BF07 Y-7982 X-FF00 A-44 B-70 C-BF S-0054	
X-FF00 0220	Modify X register contents.
>	
>RM	Display P register contents.
P-0220 Y-DEFE X-0220 A-DF B-DE C-D0 S-0054	
P-0220 (SPACE BAR)	Display remaining registers.
Y-DEFE (SPACE BAR)	
X-0220 (SPACE BAR)	
A-DF (SPACE BAR)	
B-DE (SPACE BAR)	
C-DO (SPACE BAR)	
S-0054 SPACE BAR)	(SPACE BAR) entered following stack pointer display will terminate RM command.

5.1.12 TRACE -T

T [<n>]
where: <n> is the number (in hexadecimal, $1-FF max.) of instructions to be executed. The T command allows the user to monitor program execution on an instruction-by-instruction basis. The user may optionally execute several instructions at a time by entering a count value (up to $FF). Execution starts at the current program counter (PC). The PC displays the instruction executed along with the registers values[30].

EXAMPLE	DESCRIPTION
>T	SINGLE TRACE
Op- 86	
P-0202 Y-DEFE X-FFFF A-44 B-00 C-00 S-004B	
>	MULTIPLE TRACE (2)
>T 2	
Op- B7	
P-0205 Y-DEFE X-FFFF A-44 B-00 C-00 S-004B	
Op- 01	
P-0206 Y-DEFE X-FFFF A-44 B-00 C-00 S-004B	

[30]The trace command operates by setting the OC5 interrupt to time out after the first cycle of the first opcode fetched. To activate the trace operation the OC5 pin should be connected XIRQ pin.

5.2 Interrupt vectors

Interrupt vectors residing in MCU internal Rom are accessible as follows. Each vector is assigned a three byte field residing in MONITOR memory map locations $0000-0100$. This is where the monitor program expects the MCU RAM to reside. Each vector points to a three byte field which is used as a jump table to the vector service routine. The following Table lists the interrupt vectors and associated three byte field.

INTERRUPT VECTOR	FIELD
Serial communications Interface (SCI)	$00C4-$00C6
Serial Peripheral Interface (SPI)	$00C7-$00C9
Pulse Accumulator Input Edge	$00CA-$00CC
Pulse Accumulator Overflow	$00CD-$00CF
Timer Overflow	$00D0-$00D2
Timer Output Compare 5	$00D3-$00D5
Timer Output Compare 4	$00D6-$00D8
Timer Output Compare 3	$00D9-$00DB
Timer Output Compare 2	$00DC-$00DE
Timer Output Compare 1	$00DF-$00E1
Timer Input Capture 3	$00E2-$00E4
Timer Input Capture 2	$00E5-$00E7
Timer Input Capture 1	$00E8-$00EA
Real Time Interrupt	$00EB-$00ED
IRQ	$00EE-$00FO
XIRQ	$00F1-$00F3
Software Interrupt(SWI)	$00F4-$00F6
Illegal Opcode	$00F7-$00F9
Computer Operating Properly (COP)	$00FA-$00FC
Clock Monitor	$00FD-$00FF

To use vectors specified in the table, the user must insert a jump extended opcode in the byte field of the vector required. For example, for the IRQ vector, the following is performed:
(a) place code $7E (JMP) at location $00EE and (b) place IRQ service routine address at locations $00EF and $00F0.
At IRQ interrupt the following instructions will be executed:

00EE 7E 80 00 JMP IRQ_SERVICE

5.3 Utility subroutines

Several subroutines exist that are available for performing I/O tasks. A jump table has been set up in ROM directly beneath the interrupt vectors. To use these subroutines, execute a jump to subroutine (JSR) command to the appropriate entry in the jump table. By default, all I/O performed with these routines are sent to the terminal port. Redirection of the I/O port is achieved by placing the specified value (O=SCI, 1=ACIA)into RAM location IODEV.
Utility subroutines available to the user for performing I/O tasks are:

$FFAO	UPCASE	If character in accumulator A is lower case alpha, convert to upper case.
$FFA3	WCHEK	Test character in accumulator A and return with Z bit set if character is whitespace (space, comma, tab).
$FFA6	DCHEK	Test character in accumulator A and return with Z bit set if character is delimiter (carriage return or whitespace).
$FFA9	INIT	Initialize I/O device.
$FFAC	INPUT	Read I/O device.
$FFAF	OUTPUT	Write I/O device.
$FFB2	OUTLHLF	Convert left nibble of accumulator A contents to ASCII and output to terminal port.
$FFB5	OUTRHLF	Convert right nibble of accumulator A contents to ASCII and output to terminal port.
$FFB8	OUTA	Output accumulator A ASCII character.
$FFBB	OUTlBYT	Convert binary byte at address in index register X to two ASCII characters and output. Returns address in index register X pointing to next byte.
$FFBE	OUTlBSP	Convert binary byte at address in index register X to two ASCII characters and output followed by a space. Returns address in index register
$FFCl	OUT2BSP	Convert two consecutive binary bytes starting at address in index register X to four ASCII characters and output followed by a space. Returns address in index register X pointing to next byte.
$FFC4	OUTCRLF	Output ASCII carriage return followed by a line feed.
$FFC7	OUTSTRG	Output string of ASCII bytes pointed to by address in index register X until character is na end of transmission ($04).
$FFCA	OUTSTRG0	Same as OUTSTRG except leading carriage return and line feed is skipped.
$FFCD	INCHAR	Input ASCII character to accumulator A and echo back. This routine loops until character is actually received.

The leftmost column show the addresses of utility subroutines. These subroutines can be called calling by using the 'JSR' instructions with the address of required subroutine.

5.4 S-record information

The Motorola S-record format was devised for the purpose of encoding programs or data files in a printable format for transportation between computer systems. This transportation process can therefore be monitored and the S-records can be easily edited.

5.5 S-record content

When observed, S-records are essentially character strings made of several fields which identify the record type, record length, memory address, code/data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number: the first character representing the high-order 4 bits, and the second the low-order 4 bits of the byte. Five fields which compromise an S-record are shown below:
TYPE RECORD LENGTH ADDRESS CODE/DATA CHECKSUM
where the fields are composed as follows:

FIELD	PRINTABLE CHARACTERS	CONTENTS
Type	2	S-record type -S0, S1, etc.
Record length	2	Character pair count in the record, excluding the type and record length.
Address	4,6,or 8	2-, 3-, or 4-byte address at which the data field is to be loaded into memory.
Code/data	0-n	From 0 to n bytes of executable code, memory loadable data, or descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S- record.
Checksum	2	Least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

Each record may be terminated with a CR/LF/NULL. Additionally, an S- record may have an initial field to accommodate other data such as line numbers generated by some time-sharing systems. Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

5.5.1 S-record types

Eight types of S-records have been defined to accommodate the several needs of the encoding, transportation, and decoding functions. The various Motorola upload, download, and other record transportation control programs, as well as cross assemblers, linkers, and other file-creating or debugging programs, utilize only those S-records which serve the purpose of the program. for specific information on which S-records are supported by a particular program, the user manual for that program must be consulted [31]

[31]The MONITOR monitor supports only the S1 and S9 records. All data before the first S1 record is ignored. Thereafter, all records must be S1 type until the S9 record terminates data transfer.

An S-record format may contain the following record types:

S0	Header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. The address field is normally zeroes.
S1	Code/data record and the 2-byte address at which the code/data is to reside.
S2-S8	Termination record for a block of S1 records. Address fields may optionally contain the 2-byte address of the instruction to which control is to be passed. If not specified, the first entry point specification encountered in the input will be used. There is no code/data field.

Only one termination record is used for each block of S-records. Normally, only one header record is used, although it is possible for multiple header records to occur.

5.5.2 S-record creation

S-record format programs ma be produce by several dump utilities, debuggers, or several cross assemblers or cross linkers. Several programs are available for downloading a file in S-record format from a host system to an 8-bit or 16-bit microprocessor-based system. S-RECORD EXAMPLE Shown below is a typical S-record format, as printed or displayed:

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The above format consists of an S0 header record, four S1 code/data records, and an S9 termination record. The S0 header record is comprised of the following character pairs:

S0	S-record type S0, indicating a header record.
06	Hexadecimal 06 (decimal 06), indicating six character pairs (or ASCII bytes) follow.
00	Four-character 2-byte address field, zeroes.
00	
48	
44	ASCII H, D, and R - ”HDR”.
52	
1B	Checksum of S0 record.

The first S1 code/data record is explained as follows:

S1	S-record type S1, indicating a code/data record to be loaded/verified at a 2-byte address.
13	Hexadecimal 13 (decimal 19), indicating 19 character pairs, representing 19 bytes of binary data, follow.
00	Four-character 2-byte address field; hexadecimal address 0000,
00	indicates location where the following data is to be loaded.

The next 16 character pairs are the ASCII bytes of the actual program code/data. In this assembly language example, the hexadecimal opcodes of the program are written in sequence in the code/data fields of the S1 records;

OPCODE	INSTRUCTION	
28 5F	BHCC	$0161
24 5F	BCC	$0163
22 12	BHI	$0118
22 6A	BHI	$0172
00 04 24	BRSET	0,$04,$012F
29 00	BHCS	$010D
08 23 7C	BRSET	4,$23,$018C[32]
.	.	.
.	.	.
.	.	.
2A	*checksum of the fist record*	

The second and third S1 code/data records each also contain $13 (19) character pairs and are ended with checksums 13 and 51, respectively. The fourth S1 code/data record contains 07 character pairs and has a checksum of 92. The S9 termination record is explained as follows:

S9	S-record type S9, indicating a termination record.
03	Hexadecimal 03, indicating three character pairs (3 bytes) follow.
00	Four-character 2-byte address field, zeroes.
00	
FC	Checksum of S9 record.

Each printable character in an S-record is encoded in hexadecimal (ASCII in this example) representation of the binary bits which are actually transmitted. For example, the first S1 record above is sent as shown below.

type	length	address	code/data	checksum
S 1	1 3	0 0 0 0	2 8 5 F	2 A
53 31	31 33	30 30 30 30	32 38 35 46	32 41

[32] Balance of this code is continued in the code/data fields of the remaining S1 records, and stored in memory location 0010, etc.. . .